

Student Name: Shaun James Davies

Student ID Number: 2317135

Programme: Digital Degree Apprenticeship (DDA) Computing -
Software Development

Module: Software Engineering Principles & Testing

Module Code: ACCB5030

Assignment Title: Software Engineering Design & Testing.

Submission Deadline: 27/07/2026

Final Word Count: 4482/5000

Contents

1. Introduction	3
2. Evaluation & Reflection	3
2.1 Introduction	3
2.2 Evaluate	3
2.2.1 Reflect	3
2.2.1.1 Strengths	3
2.2.1.2 Weaknesses	4
2.2.1.3 User Stories	5
2.3 Conclusion	7
3. CRC Card Design	7
3.1 Introduction	7
3.2 Evaluate	7
3.3 Conclusion	9
4. Software Testing	10
4.1 Software Testing Evaluation	10
4.1.1 Evaluate	10
4.1.2 Black & White Box Testing	10
4.1.3 Testable Code	11
4.2 Application Tests and an Investigation of Unit Testing	11
4.2.1 Black & White box Testing Examples	11
5. Unit Testing	12
6. Overall Conclusion	12
7. References	13
8. Appendix	14

1. Introduction

The aim of this assignment is to create a valid design for a Battleship game incorporating software development (SD) approaches, methodologies, design techniques and approaches for collaborative working. These areas of focus coincide with the frameworks and practices put forward from the Agile methodology. This structured design will include Agile formats of collaborative and individual work whilst recognising the importance of software testing through evaluation, examples, techniques and academic sources to fully demonstrate essential principles of test development.

2. Evaluation & Reflection

2.1 Introduction

This exercise was conducted as a part of a team to produce a fully formed plan towards a game design process for a web-based game of Battleships with additional rules, for example drop bombs and counter manoeuvres. This process was initiated as part of an Agile process to create an iterative plan that generates outcomes. Importantly, this design was created using a collaborative process in a team environment with a focus on translating gameplay rules into system design. Continuing, this design aims to develop up to, but not including, the implementation stage. This is a design only Agile plan that incorporates gameplay logic and user interaction. The key to this process is designing user stories, a story map and CRC cards to generate a design that defines system functionality from a user perspective and to solidify design decisions at a planning stage to produce strong foundations for the game. The user stories and CRC cards helped to define the features into logical sequences linked to gameplay flow and allowed the team to identify missing features.

2.2 Evaluate

2.2.1 Reflect

2.2.1.1 Strengths

This design process aimed at focusing on the player interaction rather than focusing on system implementation. This structures the design stage in line with Agile practices. By framing the user stories in the format of “as a user, I want” the team was able to place emphasis on gameplay behaviour such as coordinate guessing rather than designing with software at the focus. This improved the clarity of the game’s functional requirements as each story explicitly described an action and its intended outcome. For example, a user story of “as a user I want to aim my shots directly to where I believe the opponent to be so that I can win the game” which described a game mechanic through a behavioural level of a user which identified what the system function should include.

This allowed for a clear impact on design quality by creating a set of functional user focused plans early in the design process. The structure became more oriented towards the output of those who are playing the game. Creating a user focused design aims to reduce the likelihood of building a software implementation with missing or ineffective code.

In contrast to this, the user stories typically captured high level game mechanic and behaviour of the players. Despite developing the game with the user experience as a focus, this design element can result in simplified game mechanic designs without truly defining the logic fully.

A clear strength from this process is the element of teamwork that can easily be incorporated into the process using user stories and user mapping. This clear format used, individually and as a team, created a common process of collaboration that promoted communication.

Another clear strength of the approach is the iterative nature of the design process. The design process allowed for the team to generate ideas effectively but not in a permanent manner. This allowed the team to refine ideas over time as the development process continued. This ensured that clear mistakes or flaws in the process could be discussed and designed out of the process.

Finally, the mapping of user stories created a level of understanding across the team as it demonstrated the key aspects and flows of the game mechanics. This approach of visualising the mechanics from start to finish created a clear prioritisation of features as laid out in the multiple releases based on the cruciality of the feature. This created a design process that included the right features first instead of treating every feature equally which would deprioritise and overload the design process with low value features.

2.2.1.2 Weaknesses

A weakness of the approach is despite designing user stories and CRC cards that incorporate the design with Agile ceremonies, as a team we have developed a lack of formal modelling designs such as those used across Agile. For example, there are no UML diagrams which, if developed, can visualise the game mechanics such as system and user interactions. This creates a possibility of game logic being incorrect and increasing the risk of errors. An improvement that could be made to rectify this failure would be to create a range of diagrams that could assist in the design process such as further structural UML diagrams and state machine diagrams.

Despite a strength relating to the benefit of user stories to develop a breadth of key elements of the game mechanics focused on the user, the user stories that we developed lacked technical depth. A high-level view of user stories was generated which focused on a user perspective only. For example, the user story “as a user I want

to choose a starting position strategically so that I can win the game” benefits the design through a user centric view. However, this aspect of the design process lacks implementation detail and creates a vague starting point when the implementation begins. As the team develops upon these user stories that lack implementation detail, the development process begins to incorporate a risk of inconsistency between developers.

Continuing, an additional weakness to the approach developed by our team is that the user stories contain limited detail with the majority only focusing on an introductory sentence rather than including methods, data structures, acceptance criteria and further technical notes. These suggested improvements are incorporated across only a selection of user stories. To improve upon the user stories, these items should be considered across all user stories.

A weakness we encountered as a team was the various absences from team members which delayed the project due to responsibilities sitting with the absent members of the team. This high reliance and dependence on team dynamics risks the development of the design under time constraints. Additionally, the team fluctuated with a different level of participation upon building the designs. This resulted in varied contributions across the team. As this design project was structured with a deadline, a risk of delays, incompleteness and design imbalance was built into the design process. An improvement for this would be to initiate the project with clear accountabilities and roles assigned to each person. This will result in a more organised team with clear objectives assigned to each person in the team which will reduce the ambiguity that we faced across the design process.

2.2.1.3 User Stories

The user stories incorporated useful design elements built through the view of a user to generate a design tailored with the user experience at the focus of the design. To illustrate the benefit of the story, the team created a list of user stories combined into a user story map. User stories are a focus on features through the view of a user to carry out a particular function of the game. The team developed and included these stories to support the design process and to create features as part of the game mechanics that are intrinsically linked to what a user would experience. This supported a user centric design.

As demonstrated in Figs. 1-3, the structure of the user stories is detailed in multiple design stages. The backbone of the design is the main flow of game mechanics developed by the team. Below, further detail is structured by activities. Continuing below, the user stories are detailed attached to the activities. The design flows from left to right highlighting the main features of the game mechanics. For example, “Start game”, “take turns”, “use special actions” and “finishing the game”.

An example of team generated user stories developed that create a benefit to the story are those detailed in Figs. 4 and 5.

Fig. 4 details a user story “As an online Battleship player I want to place all of my ships on the board during a timed setup phase so that I am ready to begin gameplay without delaying the match”. This story highlights a benefit of the design that results in a user focused core gameplay function. The user benefit demonstrates that the player gains control over the ship placement and the gameplay strategy. Additionally, this feature maintains engagement through time pressure functions and feedback towards the user. This has clear functional benefit as it introduces a controlled and consistent function across all players. A crucial impact to design of this user story is that the complexity increases due to interacting components such as the timer management and state transition between game setup and game play. The structure of the acceptance criteria allows for clear testing scenarios to be generated through the acceptance criteria. This allows for future testing to include valid and invalid conditions and parameters. This requires both Black Box and White Box testing approaches. Throughout the development of the design of this feature, it can be derived that there is a strong need for this user story and attached feature. This improves the game design by pushing efficiency. Despite this, the complexity of the user story may generate substantial implementation and testing complexity meaning that the effectiveness of the user story relies on comprehensive validation and testing measures.

Fig. 5 demonstrates a key feature of this implementation of Battleships called “Finish Game”. The user story consists of “as a player I want the game to determine and present the end-of-game state when all ships of either player are destroyed so that the game can correctly conclude with a win or a loss and prevent any further gameplay actions”. The functional benefit of this story is an essential feature that contains a mechanism for detecting when the game is complete by evaluating whether all ship squares related to a player have been destroyed to ensure a valid evaluation of the winning player results in a trigger for win and loss conditions. This feature demonstrates a benefit to the user experience by communicating an outcome of the game which reduces the confusion of the user by immediately stopping further actions of the game once concluded. An important design impact can be deduced from this user story as further state management is required for a reliable ship status throughout the game such as accurate monitoring of remaining ship segments. This user story creates a testing implication which includes validation of correct end game mechanics such as win and loss which requires both Black Box and White Box testing to capture logical conditions and state transitions. In conclusion, while this story is essential for a key feature of a game mechanic a critical judgment for this user story is that its effectiveness is linked to the accurate state tracking and robust validation logic. Failure in this testing mechanic will result in an uncertainty over the win/loss outcome for the

player and possible lack of end game mechanics resulting in a loss of integrity of the application.

2.3 Conclusion

In conclusion, the design process of the game is demonstrated accurately with varied breadth of activities, user tasks and user stories. This design highlights key user stories crucial to important game mechanics. A varied use of user stories creates a game design focused on the experience of the user. However, the lack of acceptance criteria appended to the user stories is a present threat to the design process. This lack of detail and clear outputs for every feature can result in inconsistent and incomplete implementations of testing and feature code.

3. CRC Card Design

3.1 Introduction

A set of class responsibility collaboration (CRC) cards have been generated through brainstorming sessions within the team to discuss and map out the roles linked with responsibilities to describe classes in the design of the Battleship game. Interactions between classes are detailed through a collaborator section on each card. This dictates the classes and what actions they perform and are responsible for. The CRC design process is an important aspect of object-oriented design to translate user stories and game requirements into a developed diagram for future SD. The CRC cards developed with the team included Player, Coordinates, Board_Grid, TurnManager, Ship, SpecialAction, Game and ScoreManager detailed in Fig. 6. Importantly, the team generated design has proven useful for initial design discussions and early design implementations. However, the designs may not contain enough technical detail for further features of the SD life cycle. For example, coding and testing. The following evaluation will investigate the efficacy of the cards in relation to the clarity and usefulness in relation to implementation.

3.2 Evaluate

A strength produced through the CRC cards are the visualisation and clear identification of important classes. Throughout the generation of the user stories, a large breadth of game features of user interactions had been structured and organised. This system was organised into main activities. In comparison, the CRC cards helped create logical objects from the larger user story map detailing clear roles of each class. This produces an output linked with practices from object-oriented programming as abstraction and encapsulation is used here. For example, the class Game loads and tracks specific aspects of the game, controlling the gameplay flow. The Player handles user moves and abilities. The Board_Grid detects if the grid square is occupied and manages shot tracking. The Ship encapsulates details about the ship. This demonstrates abstraction and reduces team confusion through a visually less complex

system. Despite this, this details conceptual understanding of game elements rather than containing technical or coding detail. In conclusion, this results in a CRC card that generates an abstracted view of the game design but reduces its usefulness during the implementation or coding stage. Therefore, a potential improvement would be the benefits from additional design artefacts related to design supporting development.

A strength of the CRC cards is the clear outlining of responsibilities. This provides the design with logical grouping to prevent inconsistency, duplication and implementation overlap in design and coding stages. For example, the Board_Grid, the Ship and the Player. The modularity of the design improves from these cards as each card encapsulates the unique responsibilities limited to that aspect of the game mechanics. The design mechanism described allows for the team to manage aspects of the game in a clear manner. Despite this, some CRCs are poorly designed so that responsibilities remained high level limiting the usefulness in the implementation and coding stage. For example, the Game CRC lacks specific responsibilities such as controlling gameplay towards the set-up stage of the game. Including this would benefit the implementation as developers would further understand a game mechanic.

Concluding the strengths of this evaluation, a key benefit of the CRC cards was their ability to foster collaboration within the team. The visual aspect of the design stage supports discussion within the team and created an environment where each member could contribute with their shared understanding developed through the interactive process. The CRC cards generated discussion of the interaction between the classes, which created an output of further responsibilities throughout each class. In conclusion, collaboration increased using this design method which created a view of missing elements in the responsibilities that were not obviously visible prior to collaboration. To contrast, this method relied heavily on individual and team interpretation rather than an alternative systemic process which meant the resulting CRC cards could lack technical depth.

A weakness of the CRC approach is that development features such as data structures and programming detail were not generated by the approach. Despite this being a defining feature of a high-level view such as CRC cards.

Another weakness of the CRC card approach generated by the team was the lack of dynamic system behaviour detailing game mechanic in a more comprehensive way. For example, the game must respond differently depending on multiple aspects of the game mechanics such as game state, player action and ship condition. The drop bomb action in the SpecialAction card briefly touches upon this mechanic through “represent special actions” in Fig. 7. A full demonstration of this feature would include its function in a detailed manner including if a ship has been hit or if all coordinates have been hit by the other player. In summary, this limits the usefulness of the CRC cards through limited linkage to important user actions and outcomes of the game. Lacking this detail,

the implementation process includes ambiguity and can generate incorrect features and testing elements.

Continuing, as a key element of the team's design process the CRC cards lacked enough detail to inform testing design. For example, the Coordinates CRC card validates entries of those that are "within board bound" but does not specify how the invalid format is handled. Similarly, the TurnManager card can "prevent actions when it is not the player's turn" but does not include a responsibility that details how this is communicated to the user. This results in with inconsistent and uncertain testing parameters through the lack of abnormal conditions stated within the responsibilities. This generates a greater risk of invalid inputs or potentially incorrect gameplay states that generate inconsistent behaviour.

The CRC cards contain detail from the acceptance criteria identified in the user stories. For example, the CRC named game contains a responsibility linked to the finish game user story. The ability to check the win and loss conditions is linked to the "winning and losing outcomes" described in the story. Across the CRC cards there are examples like this. Despite this, a variety of user stories contain a lack of information, explicitly not containing any acceptance criteria. This reduces confidence in this planning exercise to reliably produce a product that reflects the intricate requirements of this game design.

3.3 Conclusion

In conclusion, the design for the CRC cards reflects many aspects of the game requirements identifying the main classes, responsibilities and collaborations. The responsibilities were identified and allocated effectively within the classes Game, TurnManager and Board_Grid. The game architecture and overall object interactions are clearly described although with minimum detail. Continuing, the limitations of this design are reflected from the lack of detail which can prevent a successful and full implementation that reflects user stories and usability. The cards lack a direct link to methods or functions that will later be crucial elements to the game mechanic. This results in testing concerns as they lack testing outcomes and initial drafts of logic and rules. Several user stories lack acceptance criteria which will involve the testing stage to lack comprehensiveness with the increased ambiguity. In summary, the game design lacks essential detail to improve confidence that all game requirements will be met. However, the essential game mechanics have been described through some in depth user story and CRC discussion which create a sufficient conceptual design.

4. Software Testing

4.1 Software Testing Evaluation

4.1.1 Evaluate

Software testing is a process of applying inputs to a software system to identify errors and to assess its “correctness” and effectiveness of delivering its required output. It is the “primary means of assessing software quality” for the discipline [1]. Testing has developed alongside SD as a fundamental element of the SD life cycle. The purpose of software testing is to provide factual outcomes from the logic of the software to ensure it performs as its intended purpose which can expose imperfect code prior to the release of the project. This strengthens the software quality, reliability and robustness [2]. The alternative would be a solution where essential code elements cannot be trusted to work for the intended purpose. Testing improves “trust” and confidence where multiple coding elements interact [3].

In summary, the direct link to testing and the Battleship game is that testing ensures the Battleship game plays as the structure in the design illustrates. A high-quality testing practice can produce high confidence in effectiveness [4] for the required game mechanics in the Battleship game, such as the Player, Board_Grid and ship and validates the user interactions and game state management. For example, a test in this game should confirm and validate which user wins the game and the reflected mechanics trigger correctly as detailed in user story Finish Game and CRC card Game. Additionally, a test case can be developed for the special feature Drop Bomb to ensure it damages the intended targeted ship. This improves the user experience by ensuring a fair game mechanic.

4.1.2 Black & White Box Testing

Black Box testing tests software through verifying the software’s externally observable behaviour. In practice, this requires a variety of expected inputs and outputs of game mechanics without knowledge of internal implementation [5]. Black box testing confirms the specific requirements rather than source code [6]. Continuing, Black Box testing confirms the inputs and outputs assures the acceptance criteria. Within the Battleship game this relates to the design elements of user stories and CRC cards. For example, a Black Box test for placing ships should reflect the detail in the acceptance criteria in Fig. 8. Namely, placing the ship in the given time and valid board square should result in the “ship being placed on the board”. Additionally, if the user decides to select a ship placement outside the board boundaries the placement should be rejected. These are tests that are carried out by the user and from this the required game mechanics can be validated immediately from confirmation or rejection against the game design. In summary, this type of testing is useful in validating externally observable system behaviour which testers “usually lack knowledge” about the system and its implemented code “constraints” [6].

White Box testing tests the code implementation and elements of individual code blocks, functions and methods [5]. This typically requires a software engineer, or an experienced tester or “developer” with knowledge of individual features and code structure where logic, methods, algorithms and object orientated designed features are implemented with SD [6] [7]. This less abstracted version of testing may provide a “more efficient testing” but can be “insufficient” in some cases [6]. For example, in relation to the Battleship game, a software tester will analyse the “finishing the game” feature and ensure it “executes all conditional branches” to confirm functionality. Confirming the game ends only when all ship squares are destroyed for one user. This includes a condition checking the ships are all placed and only when a final ship is removed from play the users are notified of their win and loss as described in Fig. 9.

To evaluate, Black Box testing validates that the developed software application meets the design and user requirements whilst White Box testing confirms the validity of internal code. Black Box is to investigate “fundamental aspects of the system” while White Box is “the detailed investigation of internal logic and structure of the code” [7]. The pairing of both methods of testing is sufficient to provide confidence in the code but using either technique individually can compromise the user experience in the final product. As demonstrated in Table I, this pairing differs primarily in the tester’s knowledge of the software implementation, in the level of coverage and types of defects each technique can identify. An intermediate option between the two is called Grey Box testing [7].

4.1.3 Testable Code

Code testability is vital in ensuring the software product is abiding by the logic set out in the design and ensures the code can be verified in manual or automatic testing. Testable code or “software testability is the degree to which a software system or a unit under test supports its own testing”. Code that is developed in blocks, functions and other modular components that can be tested when its inputs can be controlled and its outputs can be observed [8]. A direct example in relation to this with the Battleships game is the Drop Bomb feature. This game mechanic should be encapsulated using modularity to reduce the “impact” on other code features to reduce the risk involved with implementation. This creates the right environment for White Box testing and improves the maintainability of the software [9].

4.2 Application Tests and an Investigation of Unit Testing

4.2.1 Black & White box Testing Examples

To improve confidence in the software product, it is imperative to use both Black Box and White Box testing to validate the design and its implementation. Detailed in Table II is a variety of tests for the user story Ship Placement including the respective acceptance criteria.

5. Unit Testing

Unit testing is a software testing technique which tests units of code individually from the whole software project and is “fundamental to robust code development”. The main principles of unit testing are reviewing in small units of code to investigate through inputs and outputs throughout the development of software [10].

In relation to the ship placement tests, unit testing can be carried out using valid and invalid coordinates to confirm the placement logic accepts or rejects the inputs. The automatic placement can be unit tested to ensure the ships are placed in line with game rules. Finally, a unit test for game state transition can confirm that the Battleship game only progresses once the ship placement is in line with the game rules.

6. Overall Conclusion

The assignment aim was to use appropriate design methods for a software engineering project that incorporates software engineering principles and testing procedures to produce a Battleship game. The design process used multiple Agile techniques such as user stories, story mapping and CRC cards. This provided the design backbone for this implementation.

The design successfully identified the required game mechanics and suggested a range of features and their priority of implementation order. Core elements of this design were detailed and provided acceptance criteria that prompted a set of tests to be structured supporting the design and, ultimately, the user experience. However, a large portion of the design lacked sufficient depth and comprehensive detail which can lead to the implementation of this project failing. To strengthen the implementation and testing stages of the project, a full review of the user stories and their acceptance criteria should take place. In addition, the CRC cards will need to reflect these changes. The development of further diagrams such as UML diagrams would further strengthen the design.

In summary, the development of a software product as part of a team provided insight into industry practices and this collaborative planning and testing process creates a foundation for success from the intricate relationships between design elements and testing features, as explored in the testing examples. Despite the design lacking further detail, this design and its illustrations provide a basis on which the Battleship application would develop into a successful implementation of Agile practices, SD and testing.

7. References

- [1] A. Salahirad, G. Gay and E. Mohammadi, "Mapping the structure and evolution of software testing research over the past three decades," *The Journal of Systems & Software*, vol. 195, pp. 1-22, 2023.
- [2] F. Gurcan, "Evolution of Software Testing Strategies and Trends: Semantic Content Analysis of Software Research Corpus of the Last 40 Years," *IEEE Access*, vol. 10, pp. 106093-106095, 2022.
- [3] H. K. V. Tran, "Assessing test artifact quality—A tertiary study," *Information and Software Technology*, vol. 139, pp. 1-18, 2021.
- [4] H. K. V. Tran, "Quality attributes of test cases and test suites– importance & challenges from practitioners' perspectives," *Software Quality Journal*, vol. 33, p. 1, 2025.
- [5] K. N. HODEL, "FAT-AES: Systematic Methodology of Functional Testing for Automotive Embedded Software," *IEEE Access*, vol. 11, p. 74261, 2023.
- [6] J. Vitorino, "Constrained adversarial learning for automated software testing: a literature review," *Discover Applied Sciences*, vol. 7, no. 547, pp. 6-12, 2025.
- [7] M. E. Khan, "A Comparative Study of White Box, Black Box and Grey Box Testing Techniques," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 6, pp. 12-15, 2012.
- [8] V. Garousi, "A survey on software testability," *Information and Software Technology*, vol. 108, p. 35 (1), 2019.
- [9] M. Papamichail, "A generic methodology for early identification of non-maintainable source code components through analysis of software releases," *Information and Software Technology*, vol. 118, 2020.
- [10] H. Hunter-Zinck, "Ten simple rules on writing clean and reliable open-source scientific software," *PLOS Computational Biology*, vol. 17, no. 11, 2021.

8. Appendix

Table I – Black, Grey and White Box Testing

TABLE I. [COMPARISON BETWEEN THREE FORMS OF TESTING TECHNIQUES](#) [6] [7]

S. No.	Black Box Testing	Grey Box Testing	White Box Testing
1.	Analyses fundamental aspects only i.e. no proved edge of internal working	Partial knowledge of internal working	Full knowledge of internal working
2.	Granularity is low	Granularity is medium	Granularity is high
3.	Performed by end users and also by tester and developers (user acceptance testing)	Performed by end users and also by tester and developers (user acceptance testing)	It is performed by developers and testers
4.	Testing is based on external exceptions – internal behaviour of the program is ignored	Test design is based on high level database diagrams, data flow diagrams, internal states, knowledge of algorithm and architecture	Internal are fully known
5.	It is least exhaustive and time consuming	It is somewhere in between	Potentially most exhaustive and time consuming
6.	It can test only by trial and error method	Data domains and internal boundaries can be tested and over flow, if known	Test better: data domains and internal boundaries
7.	Not suited for algorithm testing	Not suited for algorithm testing	It is suited for algorithm testing (suited for all)

[7]

Table II – Testing Scenarios

Test	Type	Pre-Conditions	Inputs	Expected Outputs	Purpose
Valid Ship Placement	Black box	Software and game session initiated. Game mode is currently in ship movement state. Empty board squares. Timer is greater than 0.	Selecting starting coordinates that are valid.	Ship placed successfully on board.	Confirms the user can place the ship correctly. Refers to AC2.
Invalid ship placement outside of board	Black box	Ship movement state active.	Selecting starting coordinates with ship extending the board edge.	Ship placed unsuccessfully on board. Placement rejected and user message displayed.	Enforces board boundaries and that invalid board sections cannot be used. Refers to AC3.
Automatic placement of ship	White box	User has not used their ships. Empty squares are available for placement. Timer expired.	Timer expired value to return true.	Ships placed in line with game rules enforced.	Correctly return ship positions automatically. Refers to AC8
Gameplay transition	White box	All ships placed through manual or automatic intervention. Game in ship movement state.	Manual placement and automatic placement of ships.	Game state becomes Gameplay from Placement.	Prevents Gameplay beginning when game rules are conflicting. Gameplay only begins after all placement requirements are met. Refers to AC10.

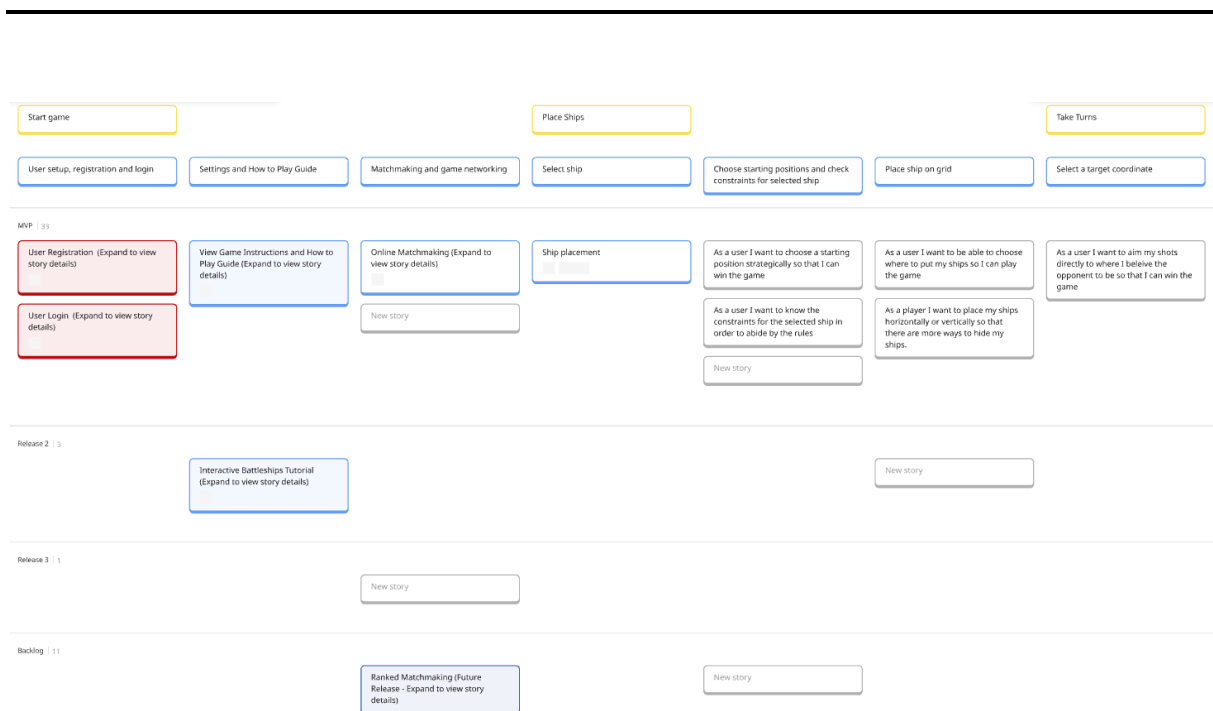


Fig. 1 – User Stories (1/3)

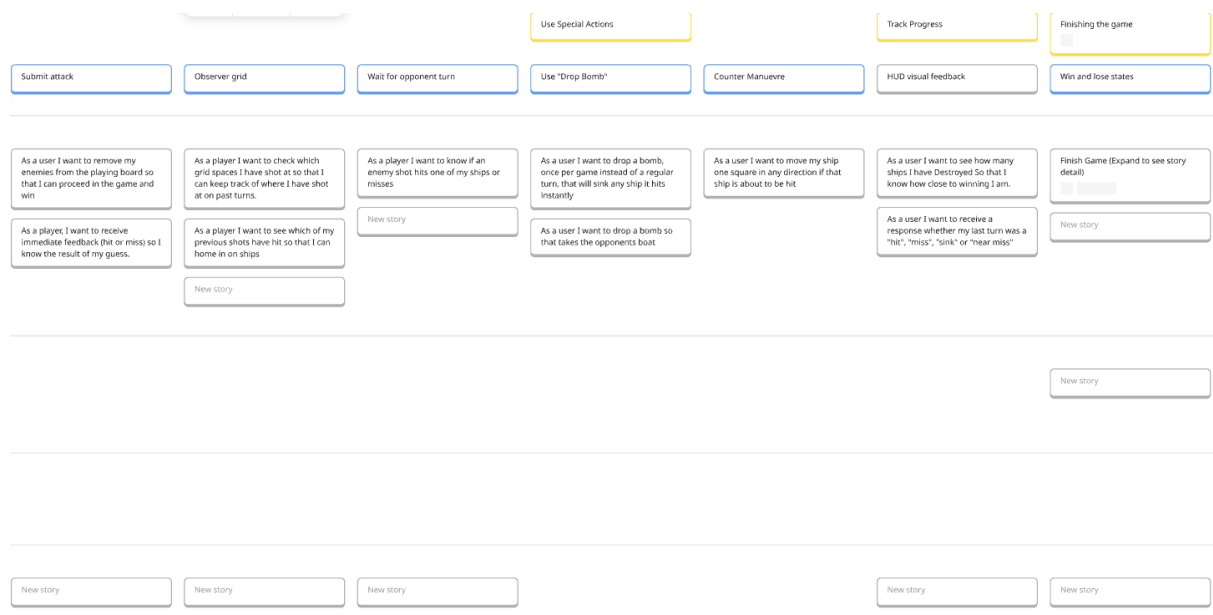


Fig. 2 – User Stories (2/3)

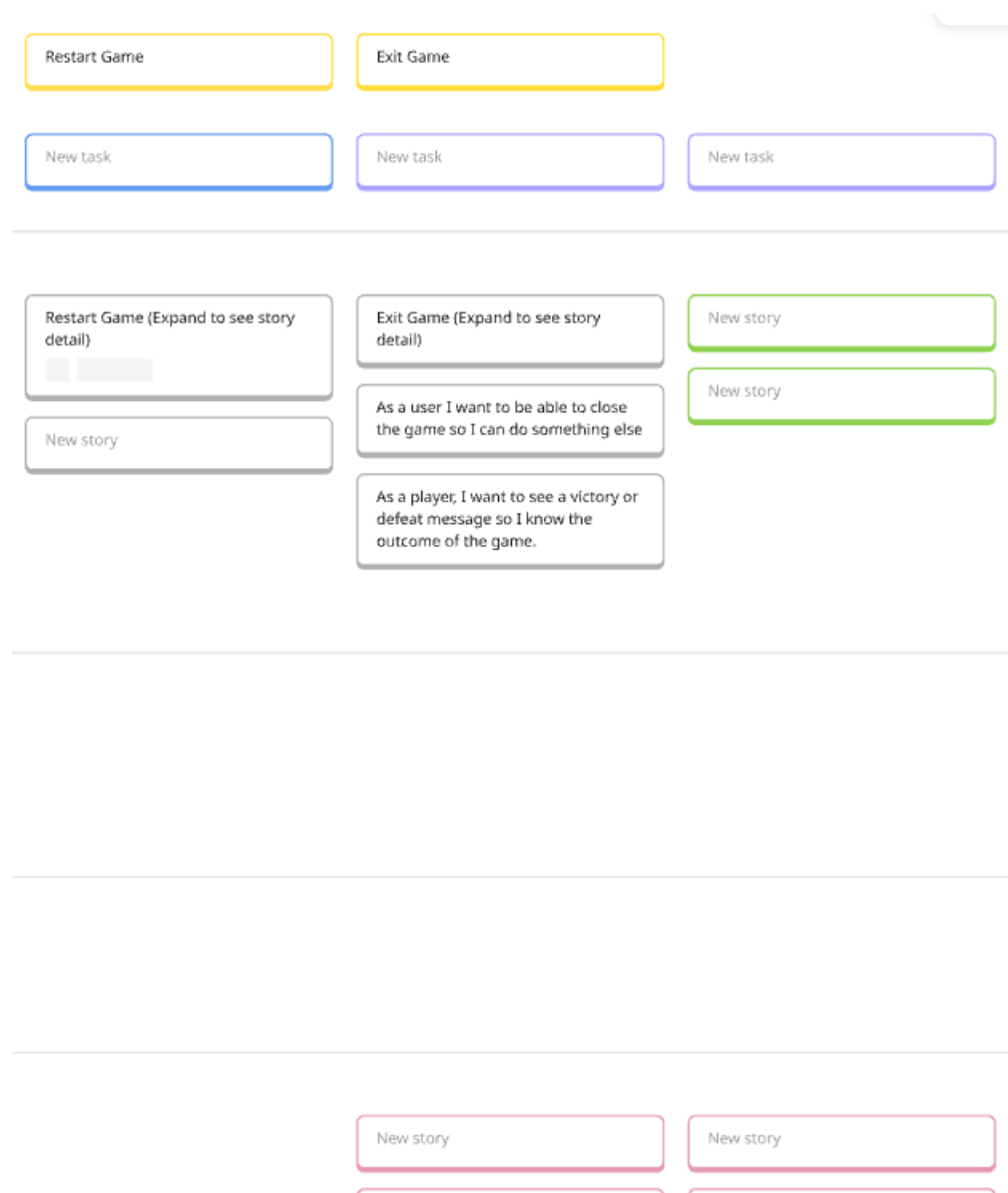


Fig. 3 - User Stories (3/3)

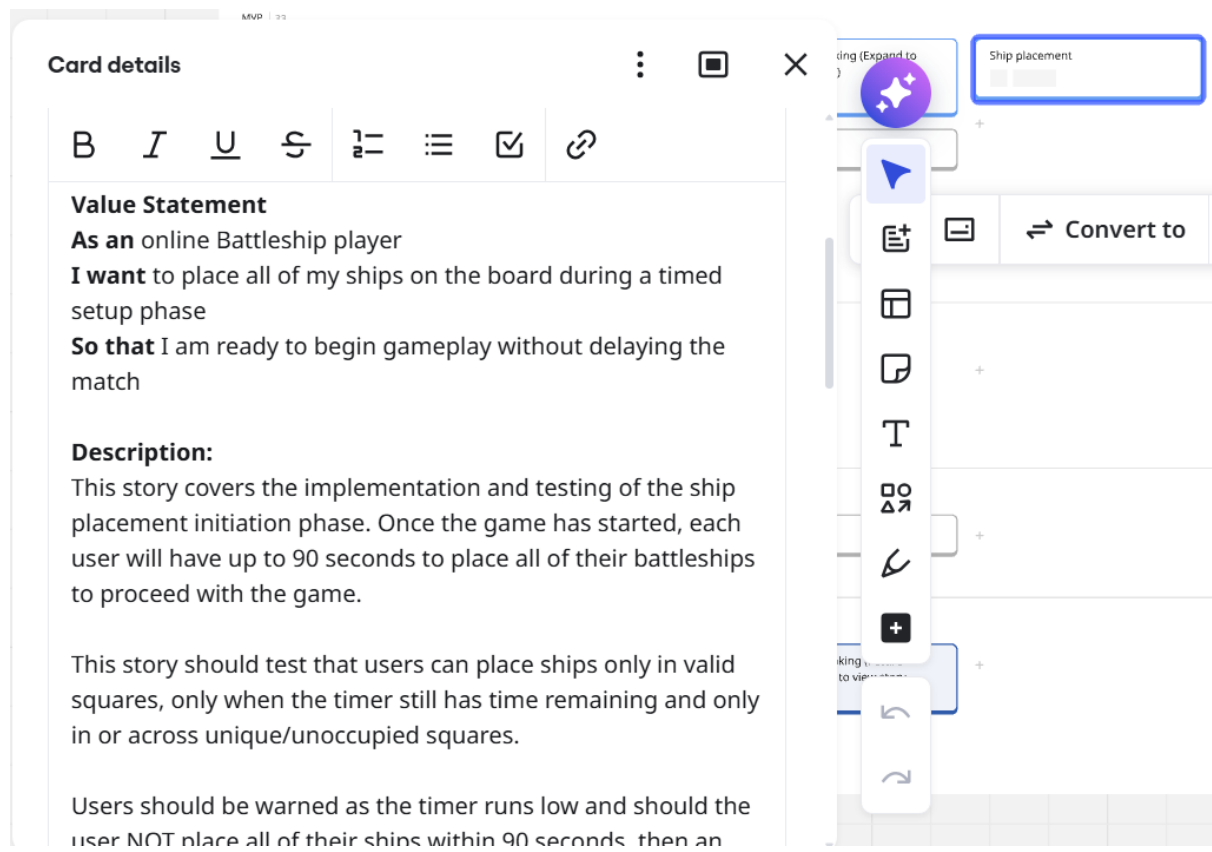


Fig. 4 – Battleship Placement

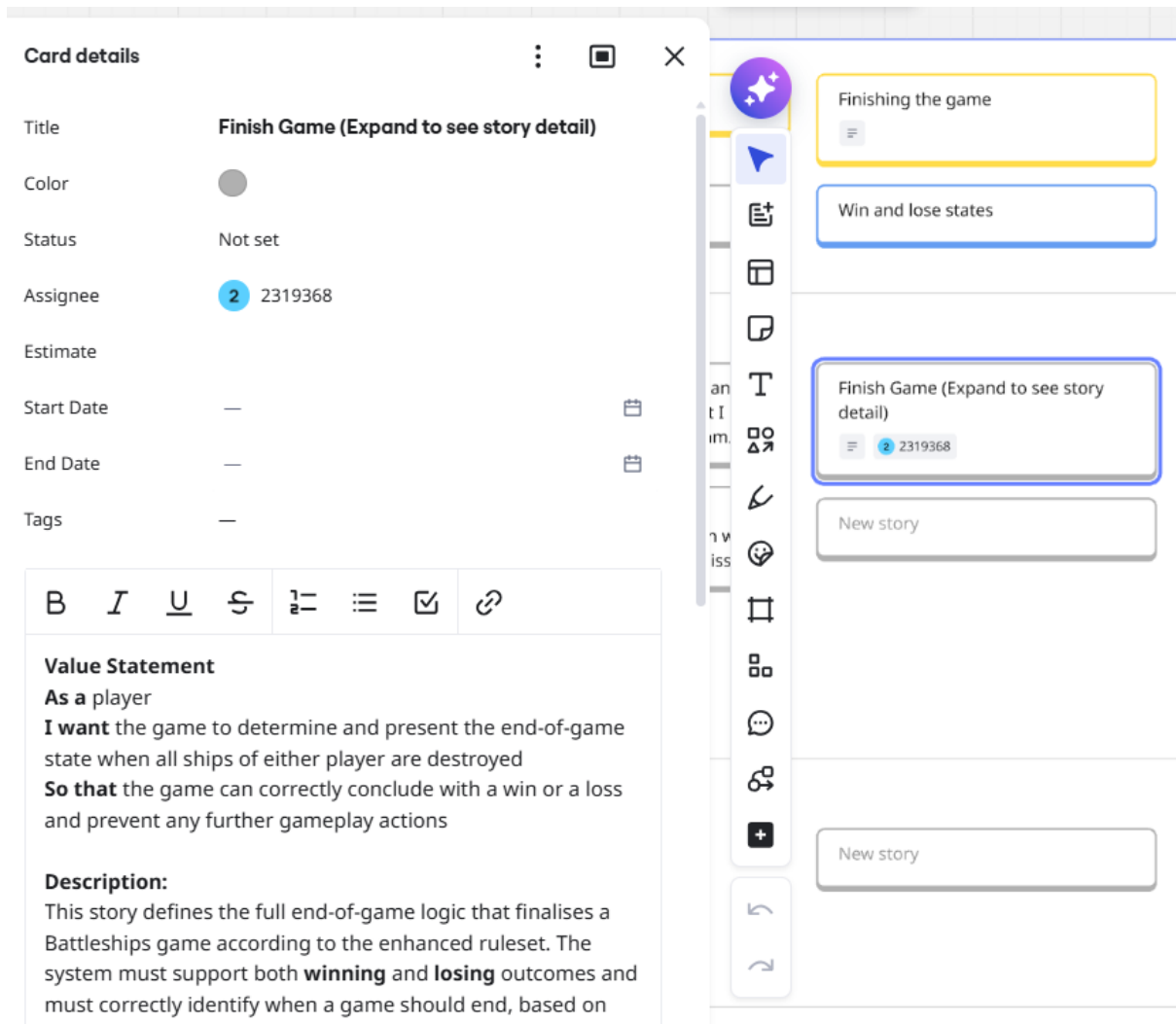


Fig. 5 – Finishing the Game

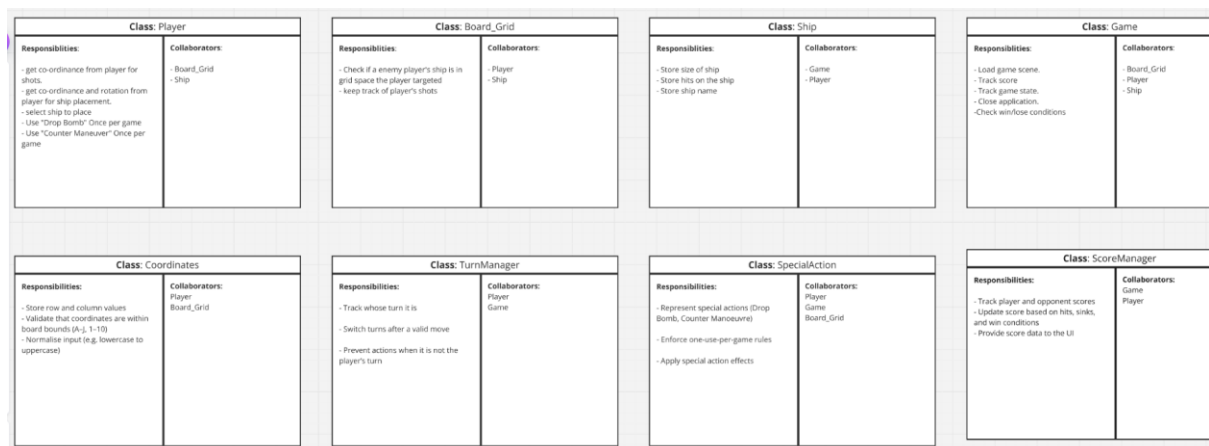


Fig. 6 – CRC cards

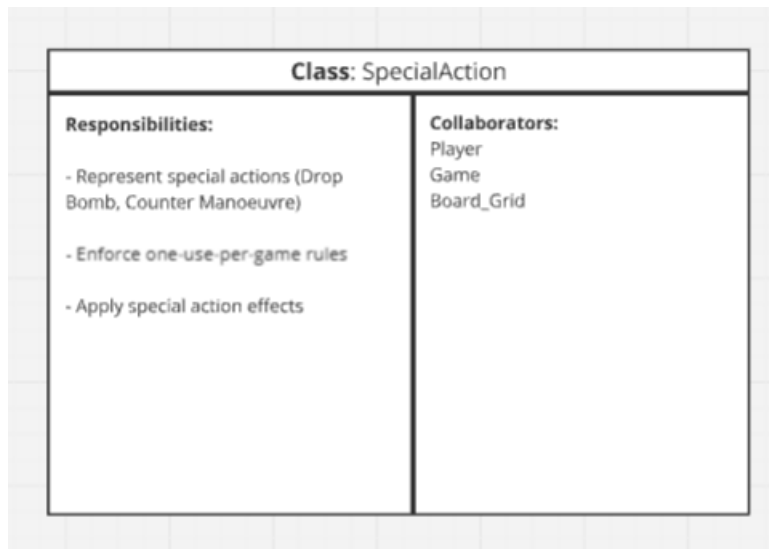


Fig. 7 – SpecialAction

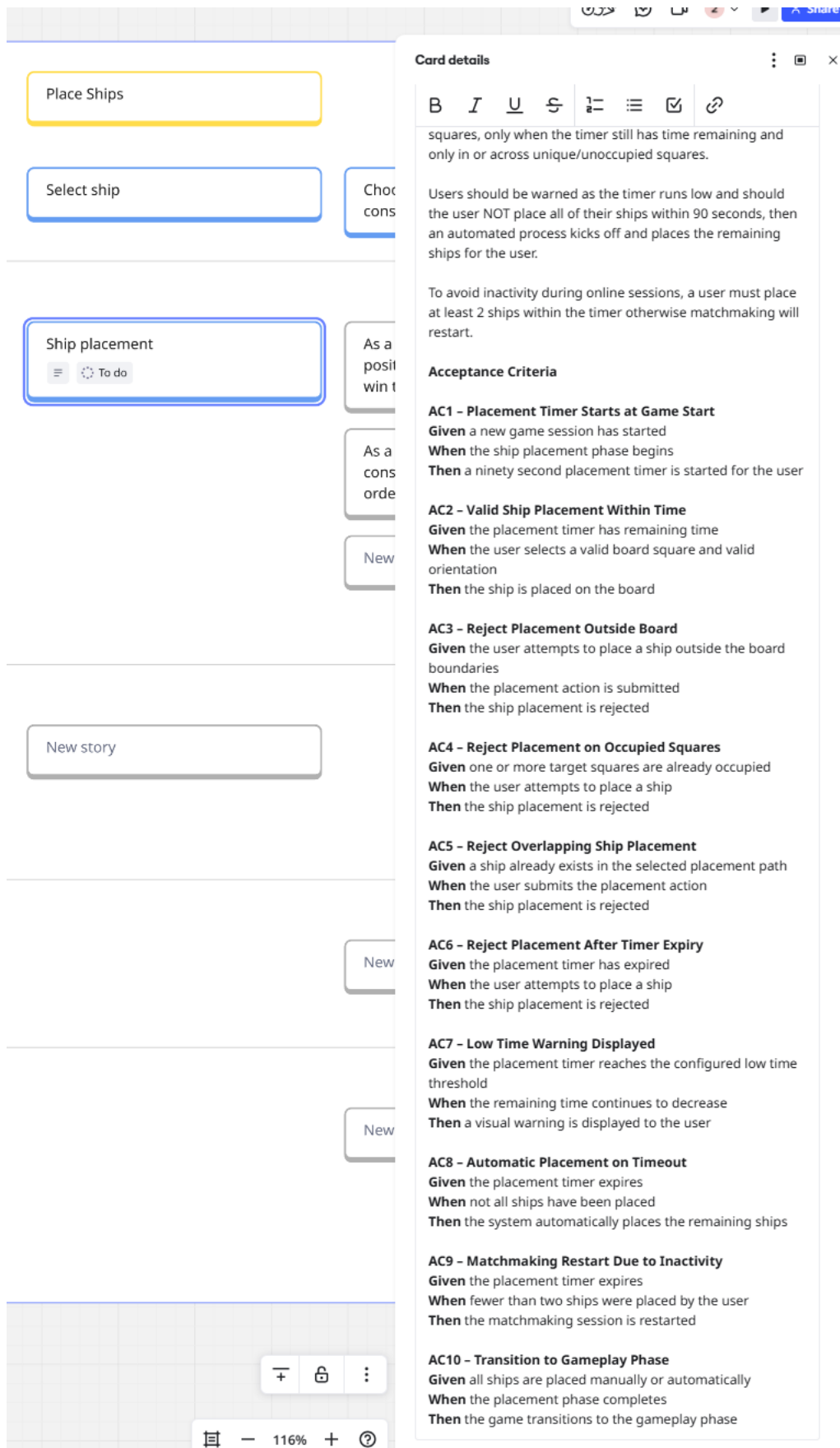


Fig. 8 – Ship Placement Acceptance Criteria

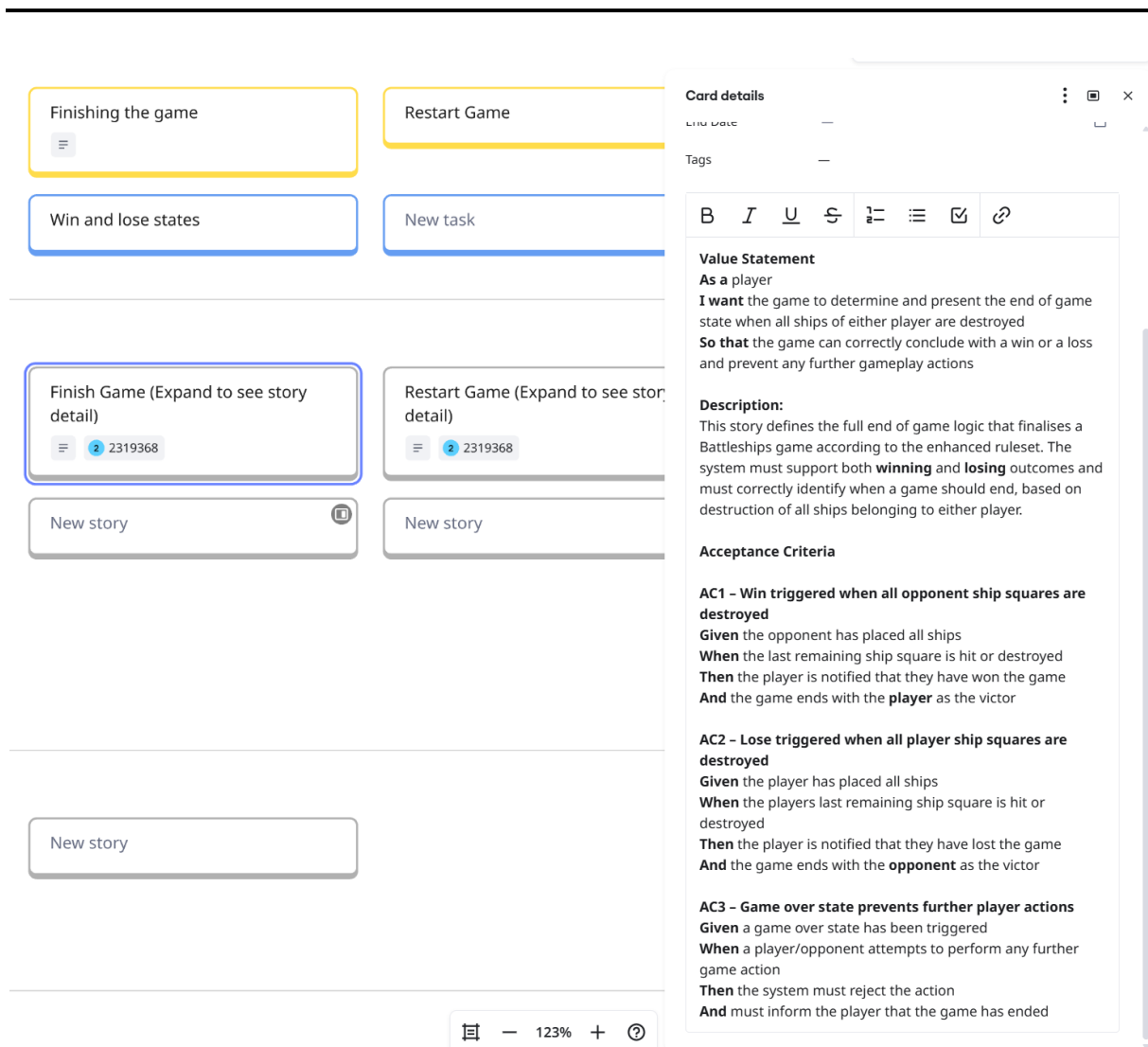


Fig. 9 – Finish Game Acceptance Criteria

Design Elements (Online link)

Merged Board - Miro https://miro.com/app/board/uXjVGwLF3rk=